Программа оценки энергии, вырабатываемой ветрогенератором за заданный интервал времени.

Исходные данные:

* Параметры ветрогенератора (рассматривать только ветрогенератор горизонтального типа – диаметр винта, кпд (КИЭВ) винта, кпд механической передачи (мультипликатора), кпд электрогенератора)
* Место расположения ветрогенератора (название населенного пункта или его широта и долгота)
* Дата и время начала и окончания заданного интервала времени

Программа должна отображать:

* Общее количество электроэнергии, выработанное ветрогенератором за заданный интервал времени
* Среднюю мощность за заданный интервал времени
* График изменения выходной мощности ветрогенератора в течение заданного интервала времени

Примечания:

* Расчеты проводить с учетом РЕАЛЬНОЙ погоды в заданной точке (населенном пункте) в течение заданного интервала времени. Информацию о реальной погоде (скорость ветра) брать из архива погоды для конкретных дат и времен (например, здесь: <https://rp5.ru/>).
* Минимальный интервал времени – 3 часа, максимальный – от заданного времени и даты начала до сегодняшнего дня.
* Естественно, что начало и окончание заданного интервала должны относиться к прошедшему времени.

Для расчетов программы курсовой работы используется ветроэнергетическая установка с горизонтальной осью вращения на рисунке 1.

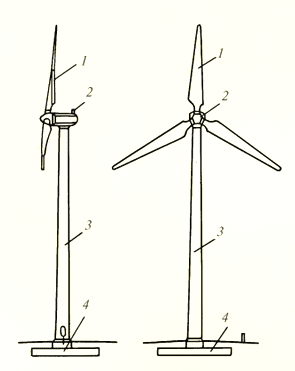


Рисунок - Ветроэнергетическая установка с горизонтальной осью вращения

Для вычислений программы в качестве примера был взят ветрогенератор WH6.4-5000W



Рисунок – ветрогенератор WH6.4-5000W

Его технические характеристики, необходимые, в качестве исходных данных:

|  |  |
| --- | --- |
| Диаметр винта, м | 6.4 |
| Коэффициент использования энергии ветра, % | 40 |
| КПД электрогенератора, % | 80% |
| КПД механической передачи, % | 90% |

Используемые формулы:

Общее количество электроэнергии, выработанное ветрогенератором за заданный интервал времени:

Где R – радиус ветроколеса, м; Vc - средняя скорость ветра, м/с; Т – расчетное время работы ветрогенератора, ч.

Средняя мощность за заданный интервал времени:

где – механическая мощность ветроколеса, кВт; – коэффициент полезного действия генератора; – КПД инвертора.

Механическая мощность ветроколеса может быть рассчитана по формуле:

где – коэффициент использования энергии ветра; – плотность воздуха(при стандартных условиях – 1.225 кг/м­3); А – площадь, ометаемая ветроколесом, м2; - скорость ветра, м/с.

Площадь, ометаемая ветроколесом может быть рассчитана как:

где – константа равная 3.14; D – диаметр винта.

Листинг основного метода программы на языке С#

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.IO;  using System.Linq;  using System.Net;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  using System.Windows.Forms.DataVisualization.Charting;  using Newtonsoft.Json.Linq;  namespace WindEnergy  {      public partial class MainForm : Form      {          private bool nonNumberEntered = false;          public MainForm()          {              InitializeComponent();          }          private void tBDiameter\_KeyDown(object sender, KeyEventArgs e)          {              nonNumberEntered = false;              if (e.KeyCode < Keys.D0 || e.KeyCode > Keys.D9)              {                  // Determine whether the keystroke is a number from the keypad.                  if (e.KeyCode < Keys.NumPad0 || e.KeyCode > Keys.NumPad9)                  {                      // Determine whether the keystroke is a backspace.                      if (e.KeyCode != Keys.Back)                      {                          // A non-numerical keystroke was pressed.                          // Set the flag to true and evaluate in KeyPress event.                          nonNumberEntered = true;                      }                  }              }              //If shift key was pressed, it's not a number.              if (Control.ModifierKeys == Keys.Shift)              {                  nonNumberEntered = true;              }          }          private void tBWPUF\_KeyDown(object sender, KeyEventArgs e)          {              nonNumberEntered = false;              if (e.KeyCode < Keys.D0 || e.KeyCode > Keys.D9)              {                  // Determine whether the keystroke is a number from the keypad.                  if (e.KeyCode < Keys.NumPad0 || e.KeyCode > Keys.NumPad9)                  {                      // Determine whether the keystroke is a backspace.                      if (e.KeyCode != Keys.Back)                      {                          // A non-numerical keystroke was pressed.                          // Set the flag to true and evaluate in KeyPress event.                          nonNumberEntered = true;                      }                  }              }              //If shift key was pressed, it's not a number.              if (Control.ModifierKeys == Keys.Shift)              {                  nonNumberEntered = true;              }          }          private void tBEfficiency\_KeyDown(object sender, KeyEventArgs e)          {              nonNumberEntered = false;              if (e.KeyCode < Keys.D0 || e.KeyCode > Keys.D9)              {                  // Determine whether the keystroke is a number from the keypad.                  if (e.KeyCode < Keys.NumPad0 || e.KeyCode > Keys.NumPad9)                  {                      // Determine whether the keystroke is a backspace.                      if (e.KeyCode != Keys.Back)                      {                          // A non-numerical keystroke was pressed.                          // Set the flag to true and evaluate in KeyPress event.                          nonNumberEntered = true;                      }                  }              }              //If shift key was pressed, it's not a number.              if (Control.ModifierKeys == Keys.Shift)              {                  nonNumberEntered = true;              }          }          private void tBInvertor\_KeyDown(object sender, KeyEventArgs e)          {              nonNumberEntered = false;              if (e.KeyCode < Keys.D0 || e.KeyCode > Keys.D9)              {                  // Determine whether the keystroke is a number from the keypad.                  if (e.KeyCode < Keys.NumPad0 || e.KeyCode > Keys.NumPad9)                  {                      // Determine whether the keystroke is a backspace.                      if (e.KeyCode != Keys.Back)                      {                          // A non-numerical keystroke was pressed.                          // Set the flag to true and evaluate in KeyPress event.                          nonNumberEntered = true;                      }                  }              }              //If shift key was pressed, it's not a number.              if (Control.ModifierKeys == Keys.Shift)              {                  nonNumberEntered = true;              }          }          private void tBTime\_KeyDown(object sender, KeyEventArgs e)          {              nonNumberEntered = false;              if (e.KeyCode < Keys.D0 || e.KeyCode > Keys.D9)              {                  // Determine whether the keystroke is a number from the keypad.                  if (e.KeyCode < Keys.NumPad0 || e.KeyCode > Keys.NumPad9)                  {                      // Determine whether the keystroke is a backspace.                      if (e.KeyCode != Keys.Back)                      {                          // A non-numerical keystroke was pressed.                          // Set the flag to true and evaluate in KeyPress event.                          nonNumberEntered = true;                      }                  }              }              //If shift key was pressed, it's not a number.              if (Control.ModifierKeys == Keys.Shift)              {                  nonNumberEntered = true;              }          }          private void tBDiameter\_KeyPress(object sender, KeyPressEventArgs e)          {              // Check for the flag being set in the KeyDown event.              if (nonNumberEntered == true)              {                  // Stop the character from being entered into the control since it is non-numerical.                  e.Handled = true;              }          }          private void tBWPUF\_KeyPress(object sender, KeyPressEventArgs e)          {              // Check for the flag being set in the KeyDown event.              if (nonNumberEntered == true)              {                  // Stop the character from being entered into the control since it is non-numerical.                  e.Handled = true;              }          }          private void tBEfficiency\_KeyPress(object sender, KeyPressEventArgs e)          {              // Check for the flag being set in the KeyDown event.              if (nonNumberEntered == true)              {                  // Stop the character from being entered into the control since it is non-numerical.                  e.Handled = true;              }          }          private void tBInvertor\_KeyPress(object sender, KeyPressEventArgs e)          {              // Check for the flag being set in the KeyDown event.              if (nonNumberEntered == true)              {                  // Stop the character from being entered into the control since it is non-numerical.                  e.Handled = true;              }          }          private void tBTime\_KeyPress(object sender, KeyPressEventArgs e)          {              // Check for the flag being set in the KeyDown event.              if (nonNumberEntered == true)              {                  // Stop the character from being entered into the control since it is non-numerical.                  e.Handled = true;              }          }          private void buttonCalculate\_Click(object sender, EventArgs e)          {              if (rBHours.Checked)              {                  if (int.Parse(tBTime.Text) >= 0 & int.Parse(tBTime.Text) <= 48)                  {                      string sTest = "";                      int i = 0;                      double[] aWinds = new double[48];                      double dWind = 0;                      string sLatitude = tBLatitude.Text;                      string sLongtitude = tBLongitude.Text;                      string sAPI = "https://api.openweathermap.org/data/2.5/onecall?lat=" + sLatitude + "&lon=" + sLongtitude + "&exclude=current,minutely,daily&appid=a3cffa38c49f0965340afea3180a025d";                      rTBTest.Text = "Подождите, идет вычисление...";                      WebRequest request = WebRequest.Create(sAPI);                      request.Method = "POST";                      request.ContentType = "application/x-www-urlencoded";                      WebResponse response = request.GetResponse();                      string answer = string.Empty;                      //получили поток ответа и записали в строковую переменную                      using (Stream s = response.GetResponseStream())                      {                          using (StreamReader reader = new StreamReader(s))                          {                              answer = reader.ReadToEnd();                          }                      }                      response.Close();                      JObject hourlyWind = JObject.Parse(answer);                      IList<JToken> results = hourlyWind["hourly"].Children().ToList();                      IList<Hourly> hourlies = new List<Hourly>();                      foreach (JToken result in results)                      {                          Hourly hourly = result.ToObject<Hourly>();                          hourlies.Add(hourly);                      }                      foreach (JToken result in results)                      {                          double inner = result["wind\_speed"].Value<double>();                          aWinds[i] = inner;                          i++;                      }                      for (int j = 0; j <= int.Parse(tBTime.Text); j++)                      {                          dWind += aWinds[int.Parse(tBTime.Text)];                      }                      double dWindAverage = dWind / int.Parse(tBTime.Text);                      double dRadius = double.Parse(tBDiameter.Text) / 2;                      double dEnergy = Math.Round((0.92 \* Math.Pow(dRadius, 2) \* Math.Pow(dWindAverage, 3) \* double.Parse(tBDiameter.Text)) / 1000, 3);                      double dPowerWT = Math.Round(double.Parse(tBEfficiency.Text) \* double.Parse(tBInverter.Text) \* 0.5 \* double.Parse(tBWPUF.Text) \* 1.225 \* (Math.Pow(double.Parse(tBDiameter.Text), 2) \* Math.PI / 4) \* Math.Pow(dWindAverage, 3) / 1000, 6);                      sTest += "Средняя скорость ветра, м/с: " + Math.Round(dWindAverage, 3).ToString() + "\nРадиус ветроколеса, м: " + dRadius.ToString() + "\nЭнергия ВЭУ, кВТ: " + dEnergy.ToString() + "\nСредняя мощность: " + dPowerWT.ToString();                      rTBTest.Text = sTest;                      Graphic newGraphic = new Graphic();                      newGraphic.Owner = this;                      newGraphic.chart1.Series[0].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Spline;                      Axis ax = new Axis();                      ax.Title = "Время, ч";                      newGraphic.chart1.ChartAreas[0].AxisX = ax;                      Axis ay = new Axis();                      ay.Title = "Мощность, кВТ";                      newGraphic.chart1.ChartAreas[0].AxisY = ay;                      for (int j = 0; j < int.Parse(tBTime.Text); j++)                      {                          newGraphic.chart1.Series[0].Points.AddXY(j, Math.Round(double.Parse(tBEfficiency.Text) \* double.Parse(tBInverter.Text) \* 0.5 \* double.Parse(tBWPUF.Text) \* 1.225 \* (Math.Pow(double.Parse(tBDiameter.Text), 2) \* Math.PI / 4) \* Math.Pow(aWinds[j], 3) / 1000, 6));                      }                      newGraphic.ShowDialog();                  }                  else                  {                      rTBTest.Text = "Попробуйте ввести время в промежутке от 1 до 47";                  }              }               else                  if(rBDays.Checked)                  {                      DateTime dateTime = monthCalendar.SelectionEnd;                      DateTime dateTime1 = monthCalendar.SelectionStart;                      string sTest = "";                      int j = 0;                      int k = int.Parse(Math.Round((dateTime - dateTime1).TotalDays \* 24).ToString());                      double[] aWinds = new double[k];                      double dWind = 0;                      string sLatitude = tBLatitude.Text;                      string sLongtitude = tBLongitude.Text;                      DateTimeOffset dDateTime = monthCalendar.SelectionStart;                      for(int i = 0; i < Math.Round((dateTime - dateTime1).TotalDays); i++)                      {                          string sDateTimeUNIX = dDateTime.ToUnixTimeSeconds().ToString();                          string sAPI = "https://api.openweathermap.org/data/2.5/onecall/timemachine?lat=" + sLatitude + "&lon=" + sLongtitude + "&dt=" + sDateTimeUNIX + "&appid=a3cffa38c49f0965340afea3180a025d";                          rTBTest.Text = "Подождите, идет вычисление...";                          //rTBTest.Text = (dateTime - dateTime1).TotalDays.ToString();                          WebRequest request = WebRequest.Create(sAPI);                          request.Method = "POST";                          request.ContentType = "application/x-www-urlencoded";                          WebResponse response = request.GetResponse();                          string answer = string.Empty;                          //получили поток ответа и записали в строковую переменную                          using (Stream s = response.GetResponseStream())                          {                              using (StreamReader reader = new StreamReader(s))                              {                                  answer = reader.ReadToEnd();                              }                          }                          response.Close();                      JObject hourlyWind = JObject.Parse(answer);                      IList<JToken> results = hourlyWind["hourly"].Children().ToList();                      IList<Hourly> hourlies = new List<Hourly>();                      foreach (JToken result in results)                      {                          Hourly hourly = result.ToObject<Hourly>();                          hourlies.Add(hourly);                      }                      foreach (JToken result in results)                          {                              double inner = result["wind\_speed"].Value<double>();                              aWinds[j] = inner;                              //rTBTest.Text += aWinds[j].ToString() + "\n";                              j++;                          }                      dDateTime = dDateTime.AddDays(1);                      }                  for (int n = 0; n < k; n++)                  {                      dWind += aWinds[n];                  }                  double dWindAverage = dWind / k;                  double dRadius = double.Parse(tBDiameter.Text) / 2;                  double dEnergy = Math.Round((0.92 \* Math.Pow(dRadius, 2) \* Math.Pow(dWindAverage, 3) \* double.Parse(tBDiameter.Text)) / 1000, 3);                  double dPowerWT = Math.Round(double.Parse(tBEfficiency.Text) \* double.Parse(tBInverter.Text) \* 0.5 \* double.Parse(tBWPUF.Text) \* 1.225 \* (Math.Pow(double.Parse(tBDiameter.Text), 2) \* Math.PI / 4) \* Math.Pow(dWindAverage, 3) / 1000, 6);                  sTest += "Средняя скорость ветра, м/с: " + Math.Round(dWindAverage, 3).ToString() + "\nРадиус ветроколеса, м: " + dRadius.ToString() + "\nЭнергия ВЭУ, кВТ: " + dEnergy.ToString() + "\nСредняя мощность: " + dPowerWT.ToString();                  rTBTest.Text = sTest;                  Graphic newGraphic = new Graphic();                  newGraphic.Owner = this;                  newGraphic.chart1.Series[0].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Spline;                  Axis ax = new Axis();                  ax.Title = "Время, ч";                  newGraphic.chart1.ChartAreas[0].AxisX = ax;                  Axis ay = new Axis();                  ay.Title = "Мощность, кВТ";                  newGraphic.chart1.ChartAreas[0].AxisY = ay;                  for (int n = 0; n < k; n++)                  {                      newGraphic.chart1.Series[0].Points.AddXY(n, Math.Round(double.Parse(tBEfficiency.Text) \* double.Parse(tBInverter.Text) \* 0.5 \* double.Parse(tBWPUF.Text) \* 1.225 \* (Math.Pow(double.Parse(tBDiameter.Text), 2) \* Math.PI / 4) \* Math.Pow(aWinds[n], 3) / 1000, 6));                  }                  newGraphic.ShowDialog();              }          }          private void MainForm\_Load(object sender, EventArgs e)          {              DateTime thisDay = DateTime.Today;              monthCalendar.MaxDate = thisDay;              monthCalendar.MinDate = thisDay.AddDays(-4);          }          private void rBHours\_CheckedChanged(object sender, EventArgs e)          {              if(rBHours.Checked)              {                  monthCalendar.Visible = false;                  tBTime.Visible = true;              }          }          private void rBDays\_CheckedChanged(object sender, EventArgs e)          {              if(rBDays.Checked)              {                  monthCalendar.Visible = true;                  tBTime.Visible = false;              }          }      }  } |

Ввод данных.

Прежде, чем приступать к рассчету в программе необходимо указать следующие данные: диаметр винта, КИЭВ, КПД генератора, КПД инвертора, ширина и долгота желаемого населенного пункта. Указать точно город нет возможности из-за способа запроса моего API ключа сервиса OpenWeather API, предоставляющего данные по скорости ветра. Также необходимо указать временной промежуток в часах или днях. Ограничения, наложенные на невозможность выбрать большее количество часов или дней (максимум часов 48, максимум дней 5) связано с особенностями предоставления доступа бесплатного API ключа к историческим данным погоды.

Для выполнения программы необходимо нажать кнопку «Рассчитать».

Результат выполнения программы:

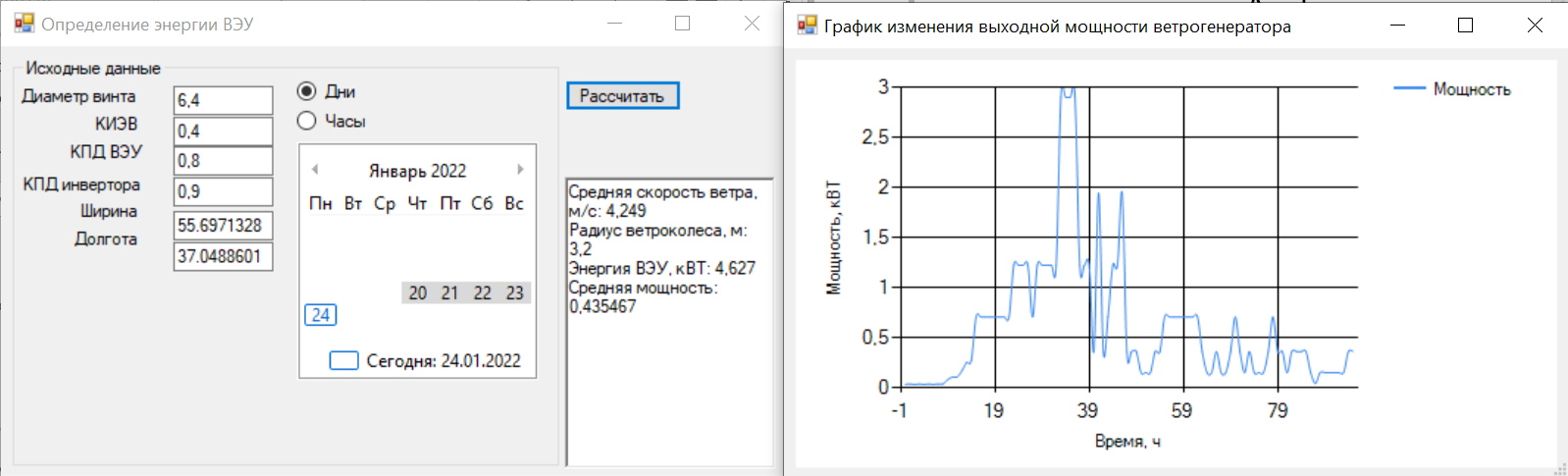


Рисунок - Результат выполнения программы по дням

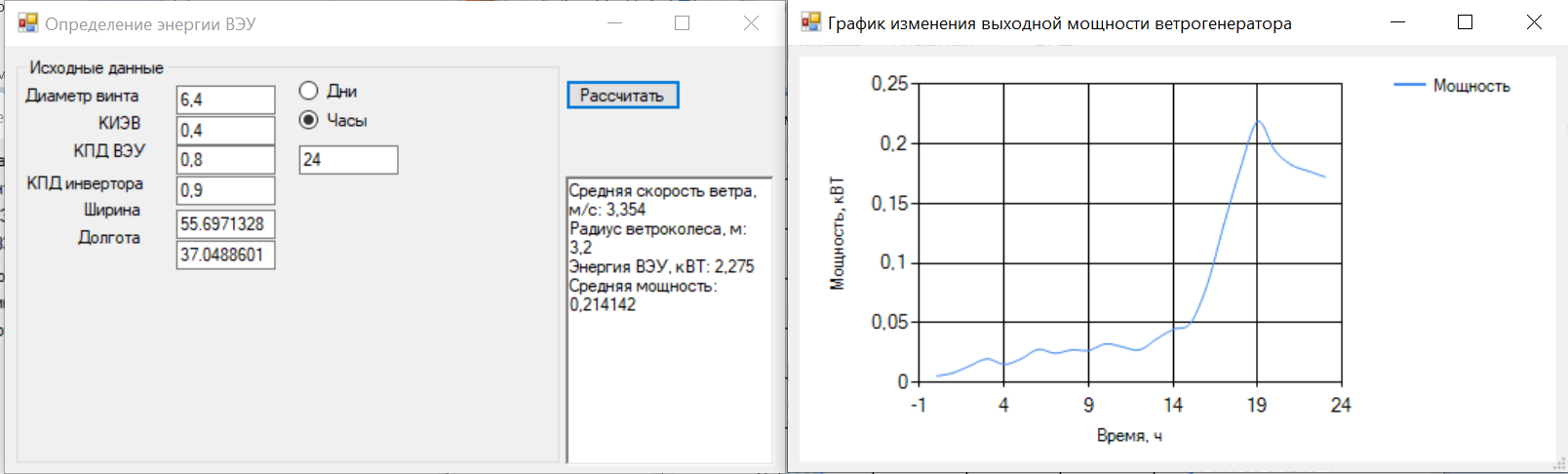


Рисунок - Результат выполнения программы по часам

Программа была выполнена в среде Visual Studio 2022, на платформе .Net Framework 4.8 с использованием языка программирования C#, сторонней библиотеки Newton JSON(<https://www.newtonsoft.com/json>) и сервисом, предоставляющим реальные данные о погоде в формате JSON OpenWeather API (https://openweathermap.org/).